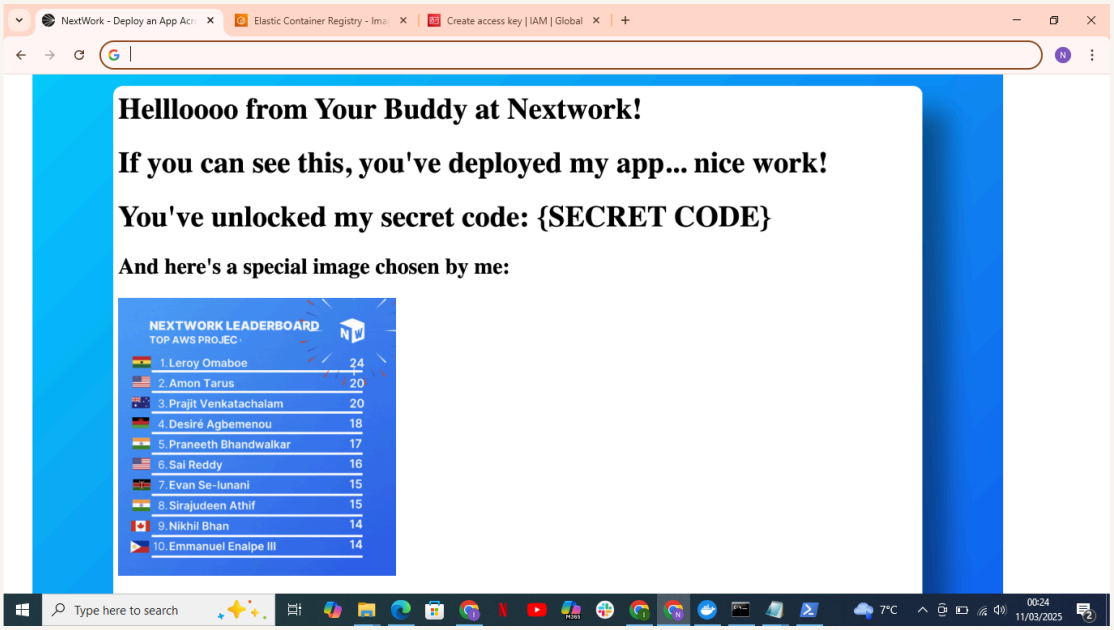
nextwork.org

Deploy an App Across Accounts

Negbe Pierre



N
Negbe Pierre
NextWork Student

Introducing Today's Project!

In this special multiplayer project, I'm working with a buddy to build, store, and share Docker container images using AWS Elastic Container Registry (ECR). Together, we are deploying applications across multiple AWS accounts, ensuring secure and efficient cross-account deployments. This involves creating Docker images for our custom applications, securely storing them in Amazon ECR, and exchanging container images with each other to launch and test applications. Through this hands-on collaboration, we are enhancing our cloud deployment skills, improving our understanding of AWS security best practices, and optimizing workflow automation for scalable cloud applications

What is Amazon ECR?

Amazon Elastic Container Registry (ECR) is a fully managed Docker container registry that makes it easy to store, manage, and deploy container images securely. In today’s project, I used ECR to store and share my container image, allowing my project buddy to pull and deploy it. I authenticated Docker with ECR, pushed my built image, resolved permission issues, and successfully deployed an application using the stored image. This ensured a seamless workflow for cross-account container deployment.

Negbe Pierre
NextWork Student

One thing I didn't expect...

One thing I didn’t expect in this project was the complexity of setting up permissions for cross-account access in Amazon ECR. While pushing and pulling images seemed straightforward, resolving permission issues required updating IAM roles, repository policies, and ensuring correct authentication. It was a great learning experience in managing secure access between AWS accounts while working with containerized applications.

This project took me...

200 minutes

N

Negbe Pierre
NextWork Student

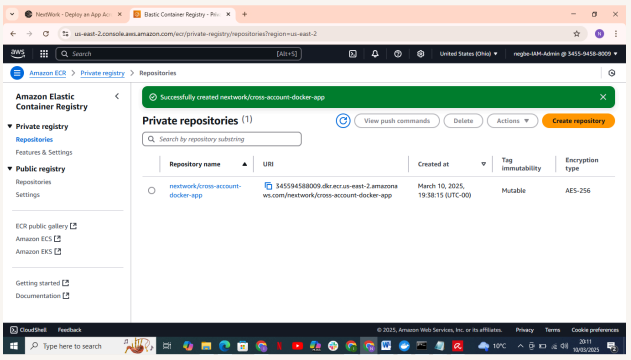
Creating a Docker Image

I set up Dockerfile and index.html. Both files are needed because: Dockerfile defines the instructions for building a Docker image using Nginx as the web server. It ensures that the container will serve the correct web content by copying the index.html file into the appropriate Nginx directory. index.html is the actual webpage content that will be displayed when the container runs. It replaces the default Nginx welcome page with a custom webpage inside the container. Together, these files allow me to create a containerized web server that serves my custom webpage

My Dockerfile tells Docker to create a containerized web server using Nginx and serve a custom HTML file. It pulls the latest Nginx image as the base (FROM nginx:latest). It copies my index.html file into the default Nginx web directory (COPY index.html /usr/share/nginx/html/). This ensures that when the container runs, Nginx serves my custom webpage instead of the default welcome page. This setup allows me to deploy a lightweight, portable web server that can be run anywhere using Docker.

I also set up an ECR repository

ECR stands for Amazon Elastic Container Registry. It is important because it provides a secure, scalable, and managed container image registry that integrates seamlessly with AWS services like ECS, EKS, and Lambda. With ECR, developers can store, manage, and deploy container images without worrying about infrastructure management, security updates, or scalability. It ensures that teams always have access to the latest container versions for efficient deployments



N
Negbe Pierre
NextWork Student

Set Up AWS CLI Access

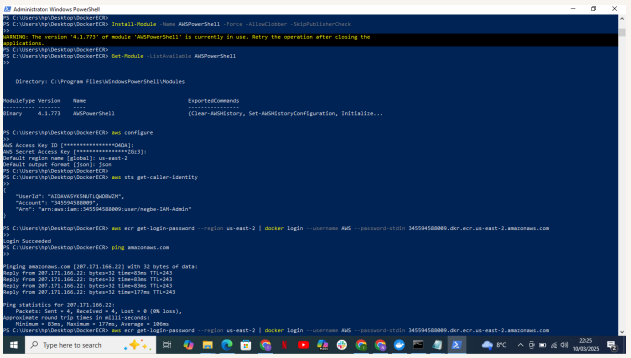
AWS CLI can let me run ECR commands

AWS CLI is the Amazon Web Services Command Line Interface, a tool that allows users to manage AWS services through commands in a terminal instead of using the AWS Management Console. The CLI asked for my credentials because it needs authentication to interact with AWS securely. By running aws configure, I provided my Access Key ID and Secret Access Key, which allow the CLI to make authorized API requests on my behalf. This setup is essential for automating AWS tasks

To enable CLI access, I set up a new IAM user with the permission to access Amazon ECR for the NextWork project on cross-account deployment. I also set up an access key for this user, which means I can authenticate programmatically to AWS services, such as ECR, using the AWS CLI instead of logging in manually. This allows me to push, pull, and manage container images securely across accounts while following AWS best practices for identity and access management.

To pass my credentials to the AWS CLI, I ran the command aws configure. I had to provide my Access Key ID, Secret Access Key, AWS Region Code (e.g., us-west-2), and the preferred output format (default is json). This step ensures that my AWS CLI is authenticated and can interact with AWS services like Amazon ECR, allowing me to push and pull container images securely without manually logging in each time.

Negbe Pierre
NextWork Student



N
Negbe Pierre
NextWork Student

Pushing My Image to ECR

Push commands are AWS CLI commands used to upload container images from a local machine to Amazon Elastic Container Registry (ECR). These commands help authenticate Docker with ECR, tag images, and push them securely to the repository. This ensures that applications can pull the latest images for deployment. The process includes: 1) Authenticate with aws ecr get-login-password. 2) Tag the image for ECR. 3) Push using docker push <ECR_URI>. This enables seamless containerized app deployment.

There are three main push commands

To authenticate Docker with my Amazon Elastic Container Registry (ECR) repository, I executed the following command:"aws ecr get-login-password --region <your-region> | docker login --username AWS --password-stdin <your-account-id>.dkr.ecr.<your-region>.amazonaws.com "This command enables secure authentication by retrieving an access token via the AWS CLI and passing it directly to Docker. By using this method, I ensured that my Docker client could interact with the ECR repository, allowing me to push and pull container images efficiently. This step is crucial for maintaining a seamless container deployment workflow within AWS.

To push my container image, I ran the command: "docker push 345594588009.dkr.ecr.us-east-2.amazonaws.com/nextwork/cross-account-docker-app:latest "Pushing means uploading my locally built Docker image to Amazon Elastic Container Registry (ECR), making it accessible for deployment. This ensures that Player B and any other authorized users can pull the latest version of the image without needing manual transfers.

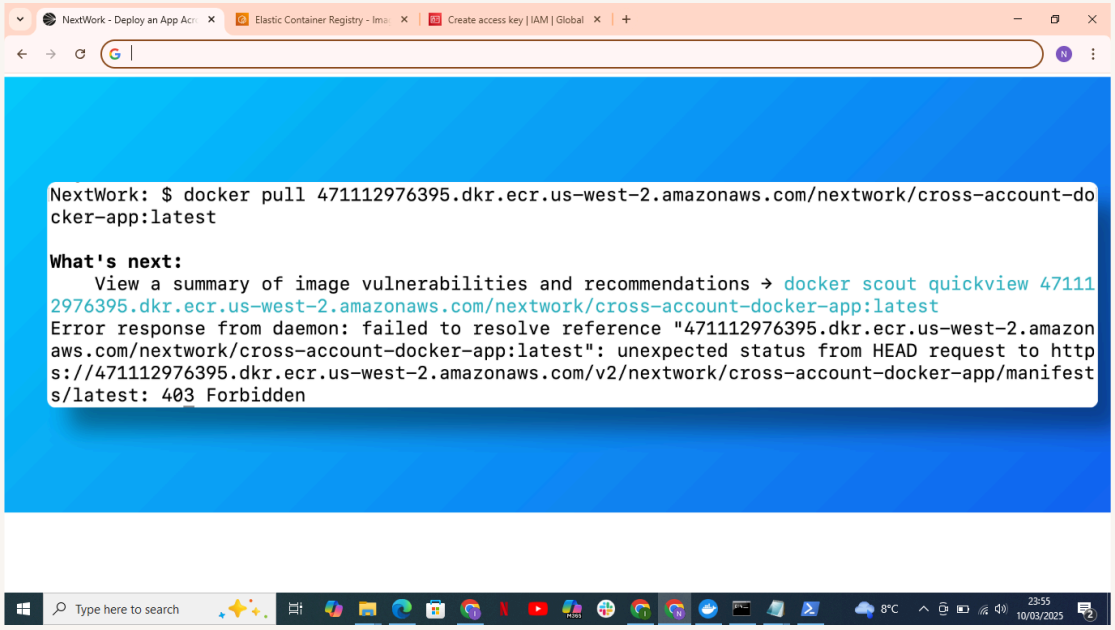
When I built my image, I tagged it with the label latest. This means that the image represents the most recent version of my application. Tagging an image as latest ensures that any deployments or services pulling from the repository will always retrieve the most up-to-date version without needing to specify a different tag manually. This approach simplifies version management and helps streamline continuous integration and deployment workflows in containerized environments.

N
Negbe Pierre
NextWork Student

Resolving Permission Issues

When I tried pulling my project buddy's container image for the first time, I saw the error "repository does not exist or may require 'docker login'". This was because I wasn't authenticated with Amazon ECR, so I needed to run the authentication command."aws ecr get-login-password --region <region> | docker login --username AWS --password-stdin <account-id>.dkr.ecr.<region>.amazonaws.com " Additionally, I was using the wrong repository URI, which meant I needed to ensure I included the full path with the image name and tag. Lastly, my buddy's repository was private, so I needed the correct permissions to access and pull their image. After fixing these issues, I successfully pulled the image

To resolve each other's permission errors, my buddy and I updated our ECR repository policies by adding each other's AWS account IDs and role ARNs. We modified the policy JSON to grant necessary pull and push permissions, ensuring seamless access and deployment.


NextWork - Deploy an App Ac
Elastic Container Registry - Ima
Create access key | IAM | Global
NextWork: $ docker pull 471112976395.dkr.ecr.us-west-2.amazonaws.com/nextwork/cross-account-docker-app:latest
What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview 471112976395.dkr.ecr.us-west-2.amazonaws.com/nextwork/cross-account-docker-app:latest
Error response from daemon: failed to resolve reference "471112976395.dkr.ecr.us-west-2.amazonaws.com/nextwork/cross-account-docker-app:latest": unexpected status from HEAD request to https://471112976395.dkr.ecr.us-west-2.amazonaws.com/v2/nextwork/cross-account-docker-app/manifests/latest: 403 Forbidden
Type here to search
8°C
23:55
10/03/2025

Everyone should be in a job they love.

Check out nextwork.org for more projects